Network Working Group
Internet-Draft
Obsoletes: 2068, 2616, 2617
(if approved)
Intended status: Standards Track
Expires: May 14, 2008

R. Fielding, Ed.
Day Software
J. Gettys
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe Systems
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT

November 11, 2007

HTTP/1.1, part 4: Conditional Requests
draft-fielding-http-p4-conditional-00

Status of this Memo

Copyright Notice

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 4 of the eight-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, updates RFC 2616 and RFC 2617. Part 4 defines request header fields for indicating conditional requests and the rules for constructing responses to those requests.

Table of Contents

1. Introduction

This document will define aspects of HTTP related to conditional request messages based on time stamps and entity-tags. Right now it only includes the extracted relevant sections of RFC 2616 [RFC2616] without edit.

2. Entity Tags

Entity tags are used for comparing two or more entities from the same requested resource. HTTP/1.1 uses entity tags in the ETag (Section 5.1), If-Match (Section 5.2), If-None-Match (Section 5.4), and If-Range ([Part 5]) header fields. The definition of how they are used and compared as cache validators is in Section 3. An entity tag consists of an opaque quoted string, possibly prefixed by a weakness indicator.

```
entity-tag = [ weak ] opaque-tag
weak       = "W/"
opaque-tag = quoted-string
```

A "strong entity tag" MAY be shared by two entities of a resource only if they are equivalent by octet equality.

A "weak entity tag," indicated by the "W/" prefix, MAY be shared by two entities of a resource only if the entities are equivalent and could be substituted for each other with no significant change in semantics. A weak entity tag can only be used for weak comparison.

An entity tag MUST be unique across all versions of all entities associated with a particular resource. A given entity tag value MAY be used for entities obtained by requests on different URIs. The use of the same entity tag value in conjunction with entities obtained by requests on different URIs does not imply the equivalence of those entities.

3. Weak and Strong Validators

Since both origin servers and caches will compare two validators to decide if they represent the same or different entities, one normally would expect that if the entity (the entity-body or any entity-headers) changes in any way, then the associated validator would change as well. If this is true, then we call this validator a "strong validator."

However, there might be cases when a server prefers to change the

validator only on semantically significant changes, and not when insignificant aspects of the entity change. A validator that does not always change when the resource changes is a "weak validator."

Entity tags are normally "strong validators," but the protocol provides a mechanism to tag an entity tag as "weak." One can think of a strong validator as one that changes whenever the bits of an entity changes, while a weak value changes whenever the meaning of an entity changes. Alternatively, one can think of a strong validator as part of an identifier for a specific entity, while a weak validator is part of an identifier for a set of semantically equivalent entities.

> Note: One example of a strong validator is an integer that is incremented in stable storage every time an entity is changed.
>
> An entity's modification time, if represented with one-second resolution, could be a weak validator, since it is possible that the resource might be modified twice during a single second.
>
> Support for weak validators is optional. However, weak validators allow for more efficient caching of equivalent objects; for example, a hit counter on a site is probably good enough if it is updated every few days or weeks, and any value during that period is likely "good enough" to be equivalent.

A "use" of a validator is either when a client generates a request and includes the validator in a validating header field, or when a server compares two validators.

Strong validators are usable in any context. Weak validators are only usable in contexts that do not depend on exact equality of an entity. For example, either kind is usable for a conditional GET of a full entity. However, only a strong validator is usable for a sub-range retrieval, since otherwise the client might end up with an internally inconsistent entity.

Clients MAY issue simple (non-subrange) GET requests with either weak validators or strong validators. Clients MUST NOT use weak validators in other forms of request.

The only function that the HTTP/1.1 protocol defines on validators is comparison. There are two validator comparison functions, depending on whether the comparison context allows the use of weak validators or not:

- The strong comparison function: in order to be considered equal, both validators MUST be identical in every way, and both MUST NOT

be weak.

- The weak comparison function: in order to be considered equal, both validators MUST be identical in every way, but either or both of them MAY be tagged as "weak" without affecting the result.

An entity tag is strong unless it is explicitly tagged as weak. Section 2 gives the syntax for entity tags.

A Last-Modified time, when used as a validator in a request, is implicitly weak unless it is possible to deduce that it is strong, using the following rules:

- The validator is being compared by an origin server to the actual current validator for the entity and,
- That origin server reliably knows that the associated entity did not change twice during the second covered by the presented validator.

or

- The validator is about to be used by a client in an If-Modified-Since or If-Unmodified-Since header, because the client has a cache entry for the associated entity, and
- That cache entry includes a Date value, which gives the time when the origin server sent the original response, and
- The presented Last-Modified time is at least 60 seconds before the Date value.

or

- The validator is being compared by an intermediate cache to the validator stored in its cache entry for the entity, and
- That cache entry includes a Date value, which gives the time when the origin server sent the original response, and
- The presented Last-Modified time is at least 60 seconds before the Date value.

This method relies on the fact that if two different responses were sent by the origin server during the same second, but both had the same Last-Modified time, then at least one of those responses would have a Date value equal to its Last-Modified time. The arbitrary 60-second limit guards against the possibility that the Date and Last-

Modified values are generated from different clocks, or at somewhat different times during the preparation of the response. An implementation MAY use a value larger than 60 seconds, if it is believed that 60 seconds is too short.

If a client wishes to perform a sub-range retrieval on a value for which it has only a Last-Modified time and no opaque validator, it MAY do this only if the Last-Modified time is strong in the sense described here.

A cache or origin server receiving a conditional request, other than a full-body GET request, MUST use the strong comparison function to evaluate the condition.

These rules allow HTTP/1.1 caches and clients to safely perform sub-range retrievals on values that have been obtained from HTTP/1.0 servers.

4. Rules for When to Use Entity Tags and Last-Modified Dates

We adopt a set of rules and recommendations for origin servers, clients, and caches regarding when various validator types ought to be used, and for what purposes.

HTTP/1.1 origin servers:

- SHOULD send an entity tag validator unless it is not feasible to generate one.
- MAY send a weak entity tag instead of a strong entity tag, if performance considerations support the use of weak entity tags, or if it is unfeasible to send a strong entity tag.
- SHOULD send a Last-Modified value if it is feasible to send one, unless the risk of a breakdown in semantic transparency that could result from using this date in an If-Modified-Since header would lead to serious problems.

In other words, the preferred behavior for an HTTP/1.1 origin server is to send both a strong entity tag and a Last-Modified value.

In order to be legal, a strong entity tag MUST change whenever the associated entity value changes in any way. A weak entity tag SHOULD change whenever the associated entity changes in a semantically significant way.

> Note: in order to provide semantically transparent caching, an origin server must avoid reusing a specific strong entity tag value for two different entities, or reusing a specific weak entity tag value for two semantically different entities. Cache entries might persist for arbitrarily long periods, regardless of expiration times, so it might be inappropriate to expect that a cache will never again attempt to validate an entry using a validator that it obtained at some point in the past.

HTTP/1.1 clients:

- If an entity tag has been provided by the origin server, MUST use that entity tag in any cache-conditional request (using If-Match or If-None-Match).

- If only a Last-Modified value has been provided by the origin server, SHOULD use that value in non-subrange cache-conditional requests (using If-Modified-Since).

- If only a Last-Modified value has been provided by an HTTP/1.0 origin server, MAY use that value in subrange cache-conditional requests (using If-Unmodified-Since:). The user agent SHOULD provide a way to disable this, in case of difficulty.

- If both an entity tag and a Last-Modified value have been provided by the origin server, SHOULD use both validators in cache-conditional requests. This allows both HTTP/1.0 and HTTP/1.1 caches to respond appropriately.

An HTTP/1.1 origin server, upon receiving a conditional request that includes both a Last-Modified date (e.g., in an If-Modified-Since or If-Unmodified-Since header field) and one or more entity tags (e.g., in an If-Match, If-None-Match, or If-Range header field) as cache validators, MUST NOT return a response status of 304 (Not Modified) unless doing so is consistent with all of the conditional header fields in the request.

An HTTP/1.1 caching proxy, upon receiving a conditional request that includes both a Last-Modified date and one or more entity tags as cache validators, MUST NOT return a locally cached response to the client unless that cached response is consistent with all of the conditional header fields in the request.

> Note: The general principle behind these rules is that HTTP/1.1 servers and clients should transmit as much non-redundant information as is available in their responses and requests. HTTP/1.1 systems receiving this information will make the most conservative assumptions about the validators they receive.

HTTP/1.0 clients and caches will ignore entity tags. Generally, last-modified values received or used by these systems will support transparent and efficient caching, and so HTTP/1.1 origin servers should provide Last-Modified values. In those rare cases where the use of a Last-Modified value as a validator by an HTTP/1.0 system could result in a serious problem, then HTTP/1.1 origin servers should not provide one.

5. Header Field Definitions

This section defines the syntax and semantics of all standard HTTP/1.1 header fields. For entity-header fields, both sender and recipient refer to either the client or the server, depending on who sends and who receives the entity.

5.1. ETag

The ETag response-header field provides the current value of the entity tag for the requested variant. The headers used with entity tags are described in sections 5.2, 5.4 and [Part 5]. The entity tag MAY be used for comparison with other entities from the same resource (see Section 3).

```
ETag = "ETag" ":" entity-tag
```

Examples:

```
ETag: "xyzzy"
ETag: W/"xyzzy"
ETag: ""
```

5.2. If-Match

The If-Match request-header field is used with a method to make it conditional. A client that has one or more entities previously obtained from the resource can verify that one of those entities is current by including a list of their associated entity tags in the If-Match header field. Entity tags are defined in Section 2. The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead. It is also used, on updating requests, to prevent inadvertent modification of the wrong version of a resource. As a special case, the value "*" matches any current entity of the resource.

```
If-Match = "If-Match" ":" ( "*" | 1#entity-tag )
```

If any of the entity tags match the entity tag of the entity that

would have been returned in the response to a similar GET request (without the If-Match header) on that resource, or if "*" is given and any current entity exists for that resource, then the server MAY perform the requested method as if the If-Match header field did not exist.

A server MUST use the strong comparison function (see Section 3) to compare the entity tags in If-Match.

If none of the entity tags match, or if "*" is given and no current entity exists, the server MUST NOT perform the requested method, and MUST return a 412 (Precondition Failed) response. This behavior is most useful when the client wants to prevent an updating method, such as PUT, from modifying a resource that has changed since the client last retrieved it.

If the request would, without the If-Match header field, result in anything other than a 2xx or 412 status, then the If-Match header MUST be ignored.

The meaning of "If-Match: *" is that the method SHOULD be performed if the representation selected by the origin server (or by a cache, possibly using the Vary mechanism, see [Part 6]) exists, and MUST NOT be performed if the representation does not exist.

A request intended to update a resource (e.g., a PUT) MAY include an If-Match header field to signal that the request method MUST NOT be applied if the entity corresponding to the If-Match value (a single entity tag) is no longer a representation of that resource. This allows the user to indicate that they do not wish the request to be successful if the resource has been changed without their knowledge. Examples:

```
If-Match: "xyzzy"
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *
```

The result of a request having both an If-Match header field and either an If-None-Match or an If-Modified-Since header fields is undefined by this specification.

5.3. If-Modified-Since

The If-Modified-Since request-header field is used with a method to make it conditional: if the requested variant has not been modified since the time specified in this field, an entity will not be returned from the server; instead, a 304 (not modified) response will be returned without any message-body.

```
If-Modified-Since = "If-Modified-Since" ":" HTTP-date
```

An example of the field is:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

A GET method with an If-Modified-Since header and no Range header requests that the identified entity be transferred only if it has been modified since the date given by the If-Modified-Since header. The algorithm for determining this includes the following cases:

1. If the request would normally result in anything other than a 200 (OK) status, or if the passed If-Modified-Since date is invalid, the response is exactly the same as for a normal GET. A date which is later than the server's current time is invalid.

2. If the variant has been modified since the If-Modified-Since date, the response is exactly the same as for a normal GET.

3. If the variant has not been modified since a valid If-Modified-Since date, the server SHOULD return a 304 (Not Modified) response.

The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead.

> Note: The Range request-header field modifies the meaning of If-Modified-Since; see [Part 5] for full details.
>
> Note: If-Modified-Since times are interpreted by the server, whose clock might not be synchronized with the client.
>
> Note: When handling an If-Modified-Since header field, some servers will use an exact date comparison function, rather than a less-than function, for deciding whether to send a 304 (Not Modified) response. To get best results when sending an If-Modified-Since header field for cache validation, clients are advised to use the exact date string received in a previous Last-Modified header field whenever possible.
>
> Note: If a client uses an arbitrary date in the If-Modified-Since header instead of a date taken from the Last-Modified header for the same request, the client should be aware of the fact that this date is interpreted in the server's understanding of time. The client should consider unsynchronized clocks and rounding problems due to the different encodings of time between the client and server. This includes the possibility of race conditions if the document has changed between the time it was first requested and

the If-Modified-Since date of a subsequent request, and the possibility of clock-skew-related problems if the If-Modified-Since date is derived from the client's clock without correction to the server's clock. Corrections for different time bases between client and server are at best approximate due to network latency.

The result of a request having both an If-Modified-Since header field and either an If-Match or an If-Unmodified-Since header fields is undefined by this specification.

5.4. If-None-Match

The If-None-Match request-header field is used with a method to make it conditional. A client that has one or more entities previously obtained from the resource can verify that none of those entities is current by including a list of their associated entity tags in the If-None-Match header field. The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead. It is also used to prevent a method (e.g. PUT) from inadvertently modifying an existing resource when the client believes that the resource does not exist.

As a special case, the value "*" matches any current entity of the resource.

```
If-None-Match = "If-None-Match" ":" ( "*" | 1#entity-tag )
```

If any of the entity tags match the entity tag of the entity that would have been returned in the response to a similar GET request (without the If-None-Match header) on that resource, or if "*" is given and any current entity exists for that resource, then the server MUST NOT perform the requested method, unless required to do so because the resource's modification date fails to match that supplied in an If-Modified-Since header field in the request. Instead, if the request method was GET or HEAD, the server SHOULD respond with a 304 (Not Modified) response, including the cache-related header fields (particularly ETag) of one of the entities that matched. For all other request methods, the server MUST respond with a status of 412 (Precondition Failed).

See Section 3 for rules on how to determine if two entities tags match. The weak comparison function can only be used with GET or HEAD requests.

If none of the entity tags match, then the server MAY perform the requested method as if the If-None-Match header field did not exist, but MUST also ignore any If-Modified-Since header field(s) in the

request. That is, if no entity tags match, then the server MUST NOT return a 304 (Not Modified) response.

If the request would, without the If-None-Match header field, result in anything other than a 2xx or 304 status, then the If-None-Match header MUST be ignored. (See Section 4 for a discussion of server behavior when both If-Modified-Since and If-None-Match appear in the same request.)

The meaning of "If-None-Match: *" is that the method MUST NOT be performed if the representation selected by the origin server (or by a cache, possibly using the Vary mechanism, see [Part 6]) exists, and SHOULD be performed if the representation does not exist. This feature is intended to be useful in preventing races between PUT operations.

Examples:

```
If-None-Match: "xyzzy"
If-None-Match: W/"xyzzy"
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-None-Match: W/"xyzzy", W/"r2d2xxxx", W/"c3piozzzz"
If-None-Match: *
```

The result of a request having both an If-None-Match header field and either an If-Match or an If-Unmodified-Since header fields is undefined by this specification.

5.5. If-Unmodified-Since

The If-Unmodified-Since request-header field is used with a method to make it conditional. If the requested resource has not been modified since the time specified in this field, the server SHOULD perform the requested operation as if the If-Unmodified-Since header were not present.

If the requested variant has been modified since the specified time, the server MUST NOT perform the requested operation, and MUST return a 412 (Precondition Failed).

```
If-Unmodified-Since = "If-Unmodified-Since" ":" HTTP-date
```

An example of the field is:

```
If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

If the request normally (i.e., without the If-Unmodified-Since header) would result in anything other than a 2xx or 412 status, the

If-Unmodified-Since header SHOULD be ignored.

If the specified date is invalid, the header is ignored.

The result of a request having both an If-Unmodified-Since header field and either an If-None-Match or an If-Modified-Since header fields is undefined by this specification.

5.6. Last-Modified

The Last-Modified entity-header field indicates the date and time at which the origin server believes the variant was last modified.

```
    Last-Modified  = "Last-Modified" ":" HTTP-date
```

An example of its use is

```
    Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

The exact meaning of this header field depends on the implementation of the origin server and the nature of the original resource. For files, it may be just the file system last-modified time. For entities with dynamically included parts, it may be the most recent of the set of last-modify times for its component parts. For database gateways, it may be the last-update time stamp of the record. For virtual objects, it may be the last time the internal state changed.

An origin server MUST NOT send a Last-Modified date which is later than the server's time of message origination. In such cases, where the resource's last modification would indicate some time in the future, the server MUST replace that date with the message origination date.

An origin server SHOULD obtain the Last-Modified value of the entity as close as possible to the time that it generates the Date value of its response. This allows a recipient to make an accurate assessment of the entity's modification time, especially if the entity changes near the time that the response is generated.

HTTP/1.1 servers SHOULD send Last-Modified whenever feasible.

6. IANA Considerations

TBD.

7. Security Considerations

No additional security considerations have been identified beyond those applicable to HTTP in general [Part 1].

8. Acknowledgments

Based on an XML translation of RFC 2616 by Julian Reschke.

9. References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

Index

Authors' Addresses

Roy T. Fielding (editor)
Day Software
23 Corporate Plaza DR, Suite 280
Newport Beach, CA 92660
USA

Phone: +1-949-706-5300
Fax: +1-949-706-5305
Email: fielding@gbiv.com
URI: http://roy.gbiv.com/

James Gettys
Hewlett-Packard Company
HP Labs, Cambridge Research Laboratory
One Cambridge Center
Cambridge, MA 02138
USA

Email: Jim.Gettys@hp.com

Jeffrey C. Mogul
Hewlett-Packard Company
HP Labs, Large Scale Systems Group
1501 Page Mill Road, MS 1177
Palo Alto, CA 94304
USA

Email: JeffMogul@acm.org

Henrik Frystyk Nielsen
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

Email: henrikn@microsoft.com

Larry Masinter
Adobe Systems, Incorporated
345 Park Ave
San Jose, CA 95110
USA

Email: LMM@acm.org
URI: http://larry.masinter.net/

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

Email: paulle@microsoft.com

Tim Berners-Lee
World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139
USA

Fax: +1 (617) 258 8682
Email: timbl@w3.org

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).